Unidad 2

Validación de Documentos XML

Apuntes realizados para la asignatura de FP Grado Superior:

Lenguajes de Marcas y Sistemas de Gestiónde Información

del ciclo Administración de Sistemas Informáticos en Red

Autor: Jorge Sánchez Asenjo (www.jorgesanchez.net)

Versión del documento: 2.0, Año 2012

Esta obra está bajo una licencia de Reconocimiento-NoComercial-CompartirIgual de CreativeCommons
Para ver una copia de esta licencia, visite: http://creativecommons.org/licenses/by-nc-sa/3.0/deed.es

creative commons

Atribución-NoComercial-CompartirIgual 3.0 Unported (CC BY-NC-SA 3.0)

Esto es un resumen fácilmente legible del Texto Legal (la licencia completa).

http://creativecommons.org/licenses/by-nc-sa/3.0/legalcode

Usted es libre de:

Compartir - copiar, distribuir, ejecutar y comunicar públicamente la obra

hacer obras derivadas

Bajo las condiciones siguientes:

Atribución — Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciante (pero no de una manera que sugiera que tiene su apoyo o que apoyan el uso que hace de su obra).

No Comercial — No puede utilizar esta obra para fines comerciales.

Compartir bajo la Misma Licencia — Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

Entendiendo que:

Renuncia — Alguna de estas condiciones puede **no aplicarse** si se obtiene el permiso del titular de los derechos de autor

Dominio Público — Cuando la obra o alguno de sus elementos se halle en el **dominio público** según la ley vigente aplicable, esta situación no quedará afectada por la licencia.

Otros derechos — Los derechos siguientes no quedan afectados por la licencia de ninguna manera:

- Los derechos derivados de **usos legítimos** u otras limitaciones reconocidas por ley no se ven afectados por lo anterior.
- Los derechos **morales** del auto;
- Derechos que pueden ostentar otras personas sobre la propia obra o su uso, como por ejemplo **derechos de imagen** o de privacidad.

índice

(2.1) documentos XML bien formados y validación _____ 7

(2.2) validación por DTD _____ 8

(2.2.1) introducción 8

(2.2.2) posibilidades de uso de DTD 8

(2.2.3) definiciones en un DTD 10

(2.2.4) elementos 10

(2.2.5) declaración de atributos 14

(2.2.6) entidades 19

(2.3) Validación por XML Schema _____ 22

(2.3.1) XML Schema y DTD 22

(2.3.2) estructura de los esquemas 22

(2.3.3) sintaxis de la definición de un elemento 27

(2.3.4) tipos simples de datos 27

(2.3.5) atributos 37

(2.3.6) tipos compuestos 37

(2.3.7) grupos de elementos 42

(2.3.8) grupos de atributos 43

(2) validación de documentos XML

(2.1) documentos XML bien formados y validación

En el tema anterior se dispusieron las bases del funcionamiento de XML. En ellas se indicó lo que se consideraba un documento XML **bien formado**. Los documentos bien formados, aseguran que las reglas de XML se cumplen y que no hay ninguna incoherencia al usar el lenguaje. Sin embargo, no es suficiente porque podríamos definir documentos que utilizaran los elementos que quisiéramos sin restricción. En la realidad los elementos y los atributos que se pueden utilizar y la manera de disponerles en el documento es fundamental para mantener una mayor homogeneidad.

Una empresa puede decidir que los documentos internos para describir el software que utiliza la empresa deben poseer como elemento raíz un elemento llamado ***software*** (y no otro) y que este elemento obligatoriamente debe contener los elementos ***nombre***, ***fabricante*** y ***precio***. Esas reglas no se refieren a que el documento esté bien formado; son reglas más complejas y que permitirán al documento que sea válido.

Para ello se crea un documento que contendrá las reglas que deben de cumplir los XML que se basen en él. De modo que un documento deberá indicar qué plantilla de reglas utiliza y deberá cumplirlas a rajatabla para considerarse válido.

Así explicado parece que la validación supone un problema, pero en realidad es una ventaja; con la validación tenemos la seguridad de que los documentos cumplen unas reglas más concretas y de esa forma es fácil establecer un protocolo en las empresas para sus documentos. De hecho cuando un documento XML cumple estrictamente las reglas generales de creación XML se dice que **está bien formado**; cuando además sigue las reglas de un documento de validación entonces se dice que es **válido**.

Las técnicas más populares para validar documentos son:

- **DTD**, ***Document Type Definition***. Validación por documentos de definición de tipos. Se utilizaba en el lenguaje SGML y de ahí debe su popularidad. Es la más utilizada, pero tiene numerosas voces críticas porque su sintaxis no es XML.
- **XML Schema** o esquemas XML. Mucho más coherente con el lenguaje XML es la aconsejada actualmente, pero todavía no tiene una implantación al 100%
- **Relax NG**. Es una notación sencilla y fácil de aprender que está haciéndose muy popular. No tiene tantas posibilidades con el XML Schema, pero tiene una sintaxis muy sencilla. Además admite añadir instrucciones de tipo XML Schema por lo que se convierte en una de las formas de validación más completas.
- **Schematron**. Permite establecer reglas que facilitan establecer las relaciones que han de cumplir los datos de un documento XML. No es tan bueno para

establecer el resto de reglas de validación (orden de elementos, tipos de datos,…)

(2.2) validación por DTD

(2.2.1) introducción

Es la técnica más veterana y, en realidad procede de SGML el lenguaje base de XML. Es indudablemente la más utilizada, pero también la menos coherente con las reglas XML. Su éxito se debe a que ya era una forma de validación reconocida antes de la aparición de XML, por lo que muchísimo productos software la reconocen desde hace mucho y la compatibilidad con ellos ha determinado su éxito.

(2.2.2) posibilidades de uso de DTD

en el propio documento

Se puede definir la estructura que debe cumplir un documento XML mediante código DTD insertado en el propio documento. La desventaja evidente, es que esta definición sólo vale para dicho documento, por lo que realmente no define tipos de documentos XML (no es una plantilla en definitiva de definición de documentos internos). Por ello es la forma menos habitual de utilizar DTD.

Su única (pero muy discutible) ventaja es que la validación está dentro del propio documento, por lo que siempre viajan juntas la validación y el contenido del mismo.

Un documento XML que defina internamente su DTD, simplemente escribe instrucciones DTD dentro del propio documento dentro de una etiqueta DOCTYPE. La sintaxis es:

```
<!DOCTYPE raíz [....códigoDTD...]>
```

Dentro de los símbolos [y] se especifican las instrucciones DTD. Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE persona [
   <!ELEMENT persona (nombre)>
   <!ELEMENT nombre (#PCDATA)>
]>
<persona>
   <nombre>Antonio</nombre>
</persona>
```

en un documento externo privado

En este caso la validación se crea en un documento-plantilla externa. De modo que cuando un documento debe cumplir las reglas de la plantilla DTD, se debe indicar la ruta (sea relativa o absoluta) a la misma.

La sintaxis de la etiqueta DOCTYPE que permite asignar un DTD privado a un documento XML es:

```
<!DOCTYPE raíz SYSTEM "rutaURLalDTD">
```

Salvo que se desee crear un único documento con una validación DTD, lo lógico es utilizar la forma de DTD externa ya que de esa forma se pueden validar varios documentos a la vez. La ruta puede ser absoluta y entonces se indica su URL:

```
<!DOCTYPE raíz SYSTEM "http://www.empresa.com/docs.dtd">
```

Pero puede ser relativa:

```
<!DOCTYPE raíz SYSTEM " docs.dtd">
```

Entonces se busca al archivo DTD desde el directorio donde se encuentra el archivo XML que queremos validar (en el ejemplo, el archivo **docs.dtd** debe encontrarse en el mismo directorio que el archivo que contiene ese código **DOCTYPE**).

En ambos casos se puede añadir código DTD para en ese documento concreto añadir instrucciones de validación. Ejemplo:

```
<!DOCTYPE raíz SYSTEM "http://www.empresa.com/docs.dtd" [
   <!ELEMENT nombre (#PCDATA)>
] >
```

DTD externo de tipo PUBLIC

Se entiende que SYSTEM se utiliza cuando el documento DTD es privado. Si se trata de un documento de uso público, entonces se usa PUBLIC. La sintaxis sería:

```
<!DOCTYPE raíz PUBLIC "nombreDTD" "DTD_URL">
```

La ***raíz*** sigue siendo el nombre del elemento raíz. El **nombreDTD** es el nombre público que se le da al DTD en cuestión. Si disponemos de un repositorio de DTDs públicos (como ocurre en entornos de trabajo como **Oxygene** por ejemplo) le cargaría sin ir a Internet. Si el **nombreDTD** no es reconocido se usa la dirección URL para descargarlo y utilizarlo. Ejemplo:

```
<!DOCTYPE html PUBLIC     "-//W3C//DTD XHTML 1.0 Strict//EN"
                "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
```

Este es el DOCTYPE para una página web escrita en XHTML 1.0 estricto, utilizada para validar miles de páginas web.

atributo standalone de la etiqueta de cabecera

La etiqueta <?xml de cabecera de todo documento XML, posee un atributo llamado standalone que puede tomar dos valores:

- **yes**. En caso de el documento XML no utilice DTD externa

- no. Cuando el documento obligatoriamente hace uso de DTD externa

(2.2.3) definiciones en un DTD

En un código DTD (tanto externo como interno) se pueden definir:

- Los **elementos** que se pueden utilizar en un documento XML. En esta definición se indica además que pueden contener dichos elementos.
- Los **atributos** que pueden poseer los elementos. Además incluso indicando sus posibles valores válidos.
- **Entidades** que puede utilizar el documento XML.

(2.2.4) elementos

Mediante un DTD podemos especificar elemento que se puede utilizar en un XML se define en su DTD mediante una etiqueta **!ELEMENT**. La sintaxis de la misma es:

```
<!ELEMENT nombre tipo>
```

El *nombre* es el identificador que tendrá el elemento en el documento XML (hay que recordar que se distingue entre mayúsculas y minúsculas).

El *tipo* indica el funcionamiento del elemento, relativo al contenido que puede tener. A continuación se indican las posibilidades de este parámetro

tipos de contenido en los elementos

EMPTY

Significa que el elemento no podrá tener contenido alguno, es un elemento vacío (como la etiqueta br de las páginas web). Ejemplo de definición de elemento vacío:

```
<!ELEMENT línea EMPTY >
```

Cuando se indica como tipo la palabra EMPTY, se indica que el elemento no puede tener contenido, pero sí podrá contener atributos (si se especifican en el DTD).

Los elementos definidos con EMPTY sólo pueden utilizarse de esta forma:

```
<línea></línea>
```

o bien:

```
<línea />
```

ANY

Permite cualquier contenido en el elemento, sin restricciones de ningún tipo. Es decir puede contener texto, otro tipo de datos y cualquier etiqueta. Además puede tener atributos.

Ejemplo:

```
<?xml version="1.0"?>
<!DOCTYPE persona [
   <!ELEMENT persona (nombre, apellidos)>
   <!ELEMENT nombre (#PCDATA)>
   <!ELEMENT apellidos ANY>
]>
<persona>
   <nombre>Jorge</nombre>
   <apellidos>Sánchez Asenjo
         <nombre>Jorge</nombre>
   </apellidos>
</persona>
```

Al definir *apellidos* como elemento ANY, permite incluso que dentro haya una etiqueta *nombre*.

Puesto que un DTD se usa para restringir la escritura de un tipo de documentos XML, el uso de ANY debe de ser muy cauteloso.

elemento concreto

En los elementos se puede indicar claramente un contenido concreto para el mismo. Dicho contenido se indica entre paréntesis. Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE persona [
   <!ELEMENT persona (nombre)>
   <!ELEMENT nombre (#PCDATA)>
]>
<persona>
   <nombre>Antonio</nombre>
</persona>
```

En el ejemplo dentro de una etiqueta *persona* **obligatoriamente** debe de existir una etiqueta *nombre* (**una y sólo una**).

No sólo se pueden indicar nombres de elementos como contenido concreto, la indicación **#PCDATA** significa que el elemento podrá contener texto literal (tan largo como se desee).

secuencias

En el caso de indicar una lista de elementos separados por comas, por ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE persona [
   <!ELEMENT persona (nombre, apellidos, edad)>
   <!ELEMENT nombre (#PCDATA)>
   <!ELEMENT apellidos (#PCDATA)>
   <!ELEMENT edad (#PCDATA)>
]>
<persona>
   <nombre>Antonio</nombre>
   <apellidos>Pérez</apellidos>
   <edad>35</edad>
</persona>
```

Indica que el elemento contendrá la lista de elementos indicada, la cual deberá estar en el mismo orden de la secuencia. Es decir en el ejemplo los *apellidos* no se podrían poner delante del *nombre*.

elecciones

Los elementos pueden contener elementos opcionales (puede aparecer uno u otro). Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE artículo [
   <!ELEMENT artículo (id | serie)>
   <!ELEMENT id (#PCDATA)>
   <!ELEMENT serie (#PCDATA)>
]>
<artículo>
   <id>16</id>
</artículo>
```

La barra vertical indica que el elemento puede contener una u otra opción (pero sólo una). Es decir también sería válido:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE artículo [
  <!ELEMENT artículo (id | serie)>
  <!ELEMENT id (#PCDATA)>
  <!ELEMENT serie (#PCDATA)>
]>
<artículo>
   <serie>X1238H</serie>
</artículo>
```

Si dentro de la lista de opciones aparece **PCDATA**, éste debe de ser el primer elemento de la lista.

combinaciones

Por supuesto puede haber combinaciones, si tenemos un documento DTD llamado *coordenada.dtd* con este contenido:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT coordenada ((longitud, latitud) | coordUniversal)>
<!ELEMENT longitud (#PCDATA)>
<!ELEMENT latitud (#PCDATA)>
<!ELEMENT coordUniversal (#PCDATA)>
```

Sería válido este documento:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE coordenada SYSTEM "coordenada.dtd">
<coordenada>
   <longitud>234</longitud>
   <latitud>-23</latitud>
</coordenada>
```

Sería válido también:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE coordenada SYSTEM "coordenada.dtd">
<coordenada>
   <coordUniversal>1232332</coordUniversal>
</coordenada>
```

cardinalidad

La cardinalidad es el número de veces que puede aparecer un determinado contenido en un elemento. Se realiza mediante estos símbolos:

- **?** Contenido opcional, puede aparecer (una sola vez) o no aparecer
- ***** Contenido opcional y repetible. Es decir puede no aparecer y puede incluso aparecer varias veces
- **+** Contenido obligatorio y repetible. Tiene que aparecer e incluso puede aparecer varias veces

Ejemplo:

```
<!ELEMENT película (título, dirección+, argumento?, actor*)>
```

Según la instrucción anterior el elemento película consta de un título, uno o más elementos de dirección, puede o no tener argumento, y de varios a ningún actor (además se tendría que respetar ese orden).

Otro ejemplo (*polígono.dtd*):

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT polígono ((coordX,coordY)+ | nombre)>
<!ELEMENT coordX (#PCDATA)>
<!ELEMENT coordY (#PCDATA)>
<!ELEMENT nombre (#PCDATA)>
```

Con esa DTD sería válido el documento:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE polígono SYSTEM "polígono.dtd">
<polígono>
   <coordX>12</coordX>
   <coordY>13</coordY>
   <coordX>17</coordX>
   <coordY>23</coordY>
   <coordX>34</coordX>
   <coordY>56</coordY>
</polígono>
```

Pero también:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE polígono SYSTEM "polígono.dtd">
<polígono>
   <nombre>Triángulo</nombre>
</polígono>
```

(2.2.5) declaración de atributos

Los atributos permiten añadir información a un elemento. Un atributo no puede constar de más atributos y cada atributo sólo puede aparecer una vez en cada elemento.

sintaxis de la declaración de atributos

Los atributos se declaran mediante la etiqueta !ATTLIST, sintaxis:

```
<!ATTLIST elemento nombreAtributo tipo presencia valorPorDefecto>
```

Donde:

- *elemento*. Es el nombre del elemento que podrá utilizar el atributo
- *nombreAtributo*. Es el identificador del atributo que estamos declarando (y que debe de cumplir las reglas de identificadores de XML)
- *tipo*. Es el tipo de valores que podemos asignar al atributo
- *presencia*. Indica las características de los valores que puede tomar el atributo: si es obligatorio, si hay valor por defecto,...

- *valorPorDefecto*. Permite dar un valor que el atributo tomará en el documento XML en caso de que no se le dé en el mismo ningún valor al atributo. También indica si es necesario rellenar o no el atributo o bien si es opcional.

CDATA

Para indicar el tipo de valores de un atributo se usa la palabra **CDATA** para indicar que el atributos contiene texto (CDATA es el acrónimo de *Character DATA*).

A diferencia de **#PCDATA**, su contenido no es procesado, lo que significa que puede contener cualquier valor (incluidos símbolos prohibidos en los #PCDATA como **<**, **>**, **&**)

declarar atributos

Esta declaración:

```
<!ATTLIST persona nacionalidad CDATA>
```

Significa que hemos definido el atributo *nacionalidad* correspondiente al elemento *persona*. Que será de tipo *CDATA*, es decir texto normal. Así en un XML que se valide con el DTD en el que está la instrucción anterior, podremos:

```
<persona nacionalidad=”española”>
```

valores por defecto

Al declarar un atributo, lo último que se indica es la propiedad relativa al valor por defecto del atributo. Se comentan a continuación sus posibilidades:

valor por defecto concreto

Si al final de la declaración de un atributo aparece un valor concreto, se entiende que ese será el valor por defecto. Es decir que se podría no utilizar el atributo en un elemento y entonces dicho atributo tomaría dicho valor.

Por ejemplo supongamos que éste es el archivo *directorio.dtd*:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT  directorio (persona)+>
<!ELEMENT  persona (#PCDATA)>
<!ATTLIST persona nacionalidad CDATA "Española">
```

Se define en él el atributo *nacionalidad* para el elemento *persona* como un atributo que contendrá texto de todo tipo, pero que por defecto toma el valor *Española* (nacionalidad por defecto en dicho archivo).

Entonces este archivo XML será válido:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE directorio SYSTEM "directorio.dtd">
<directorio>
   <persona nacionalidad="Francesa">Vivian Maret</persona>
   <persona>Juan Martín</persona>
</directorio>
```

Entonces para *Vivian Maret* se ha indicado explícitamente la *nacionalidad*, pero no se ha usado el atributo para *Juan Martín*, por lo que tomará la nacionalidad española.

valores fijos

Se puede utilizar el término #FIXED antes de indicar un valor fijo paraun atributo. En ese caso en ningún documento XML se podrá modificar dicho atributo. Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT directorio (persona)+>
<!ELEMENT persona (#PCDATA)>
<!ATTLIST persona nacionalidad CDATA #FIXED "Española">
```

El atributo *nacionalidad* no podrá tomar ningún valor que no sea el valor *Española*, impidiendo tomar otra nacionalidad. En la práctica este tipo de atributos no se usa demasiado, sólo se usa en el caso de que la presencia o no de dicho atributo en un elemento sea determinante. Es el caso del atributo noshade del elemento hr del lenguaje HTML. hr sirve para dibujar una línea en una página web, la línea se dibuja en relieve salvo que aparezca el atributo *noshade*, ya que hace que la línea sea plana. Por eso a *noshade* sólo se le puede dar un valor: la propia palabra *noshade*:

```
<hr noshade="noshade" />
```

valores requeridos

En este caso se usa la palabra #REQUIRED indicando con ello que siempre hay que dar valor al atributo. Ejemplo:

```
<!ATTLIST persona nacionalidad CDATA #REQUIRED>
```

Un documento XML que utilice el elemento *persona* deberá especificar obligatoriamente la *nacionalidad*.

Obviamente usando #REQUIRED no se puede indicar un valor por defecto (al no poder dejarse sin especificar el atributo).

valor opcional

La palabra #IMPLIED especificada en el atributo indicaría que dicho atributo puede quedarse sin valor; es decir no posee valor por defecto, pero puede quedarse sin especificar (quedaría nulo por tanto).

```
<!ATTLIST persona nacionalidad CDATA #IMPLIED>
```

En el ejemplo, el atributo nacionalidad no es obligatorio especificarle, puede quedar sin valor. Nuevamente en este caso no se puede especificar un valor por defecto (sería absurdo).

tipos de atributo

CDATA

Como se comentó antes, los atributos de tipo CDATA permiten indicar como valor cualquier texto. A diferencia de los datos PCDATA de los elementos, los CDATA admiten cualquier carácter del tipo que sea.

ID

Sirve para generar identificadores a los elementos. Un identificador es un valor único que tendrá cada elemento y son muy usados en XML. El valor de un atributo de tipo ID cumple estas reglas:

- El valor tiene que cumplir las mismas reglas que para especificar nombres XML. Es decir: nada de espacios, no pueden comenzar con un número y sólo admite letras, números y el carácter de subrayado (_).
- No puede haber dos elementos con el mismo ID en un mismo documento XML
- En el DTD, para cada elemento sólo puede indicarse un atributo como ID. No puede haber dos atributos distintos en el mismo elemento que contengan IDs.
- Los atributos ID sólo pueden indicar **#IMPLIED** o **#REQUIRED** en el apartado del valor por defecto.

Los IDs son especialmente útiles para las herramientas de maquetación, análisis y programación de aplicaciones XML. Ya que permiten diferenciar de manera única a cada elemento.

IDREF

Los atributos IDREF contienen el valor de un ID de otro elemento. Es decir será una referencia a otro elemento. Las reglas de los IDREFs son:

- El valor de un IDREF debe cumplir las reglas para especificar nombres XML (es lógico ya que contienen valores de tipo ID)
- Debe existir un atributo ID en el documento XML cuyo valor coincida con el especificado en un IDREF (de otro modo se haría referencia a un elemento inexistente y esto no está permitido)

La idea es poder relacionar elementos a través de atributos de tipo ID e IDREF. Ejemplo de uso:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Archivo directorio.dtd -->
<!ELEMENT directorio (persona)+ >
<!ELEMENT persona (#PCDATA) >
<!ATTLIST persona id ID #REQUIRED
                  madre IDREF #IMPLIED
                  padre IDREF #IMPLIED>

<?xml version="1.0" encoding="UTF-8"?>
<!-- Archivo directorio1.xml-->
<!DOCTYPE directorio SYSTEM "directorio.dtd">
<directorio>
   <persona id="p1">Pedro</persona>
   <persona id="p2">Marisa</persona>
   <persona id="p3" madre="p2" padre="p1">Carmen</persona>
</directorio>
```

Carmen es la hija de Pedro y Marisa, según el código anterior, ya que los atributos *padre* y *madre* de tipo IDREF contienen los ID de *Pedro* y *Marisa*.

IDREFS

Igual que el anterior sólo que permite indicar varias referencias (que deben existir en el documento XML) a otros ID, separadas por espacios. Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Archivo directorio.dtd -->
<!ELEMENT directorio (persona)+ >
<!ELEMENT persona (#PCDATA) >
<!ATTLIST persona id ID #REQUIRED
                  padres IDREFS #IMPLIED >

<?xml version="1.0" encoding="UTF-8"?>
<!-- Archivo directorio1.xml-->
<!DOCTYPE directorio SYSTEM "directorio.dtd">
<directorio>
   <persona id="p1">Pedro</persona>
   <persona id="p2">Marisa</persona>
   <persona id="p3" padres="p1 p2">Carmen</persona>
</directorio>
```

NMTOKEN

El valor del atributo será un texto que cumplirá las reglas para nombres XML. Se usa en atributos donde se entiende que CDATA permite demasiadas libertades, de esta forma al menos sabremos que es un texto donde sólo existirán letras, números y el símbolo _, es decir un texto que cumple las reglas para nombres XML.

NMTOKENS

El atributo puede contener varios valores de tipo NMTOKEN separados por espacios

ENTITY

El valor de un atributo será una entidad de la cual se indica el nombre. Más adelante se explica el uso de las entidades.

ENTITIES

El valor del atributo será una lista de nombres de entidades separadas por espacios.

enumeración

En este caso el valor del atributo debe de ser uno de una lista de valores posibles cada uno de los cuales se separa del siguiente mediante el símbolo |.

Ejemplo:

```
<!ATTLIST persona sexo (Hombre | Mujer) #REQUIRED >
```

Las personas sólo podrán especificar como sexo *"Hombre"* o *"Mujer"* y nada más:

```
<persona sexo="Varón">Javier Ruiz</persona>
```

El código anterior XML fallaría ya que el atributo sexo no admite el valor *"Varón"*.

declaración de varios atributos en la misma etiqueta

Se usa muy habitualmente para indicar de forma cómoda todos los atributos de un determinado elemento:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT  directorio (persona)+>
<!ELEMENT  persona (#PCDATA)>
<!ATTLIST persona   nacionalidad CDATA "Española"
                    sexo (Hombre | Mujer) #IMPLIED
                    id ID #REQUIRED>
```

Para ese documento DTD, sería válido este XML:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE directorio SYSTEM "directorio.dtd">
<directorio>
  <persona nacionalidad="Francesa" id="A1234">Vivian Maret</persona>
  <persona  id="A789">Juan Martín</persona>
  <persona sexo="Hombre" id="A12">Rafael Díaz</persona>
</directorio>
```

(2.2.6) entidades

Las entidades son elementos XML que permiten indicar abreviaturas de texto (o referencias a elementos externos abreviadas) o utilizar caracteres que de otra forma serían inválidos en el documento.

Se explican sus posibilidades

entidades ya existentes

En XML están definidas las siguientes entidades:

entidad	significado
<	El símbolo de menor (<)
>	El símbolo de mayor (>)
&	El ampersand: &
'	La comilla simple (')
"	La comilla doble (")

Estas entidades no hay que declararlas en ningún DTD, todos los analizadores de XML estándar conocen estas entidades

Ejemplo:

```
<autor>Leopoldo Alas &apos;Clarín&apos;</autor>
```

El texto PCDATA del autor es *Leopoldo Alas 'Clarín'* (así se visualizará en el navegador).

entidades para referencias a caracteres especiales

La etiqueta inicial <?xml permite indicar (entre otras cosas) el juego de caracteres que utiliza un documento XML (normalmente Unicode, UTF8).

Si deseamos indicar un carácter especial que no está contenido en nuestro teclado, conociendo su código en el juego de caracteres que utiliza el documento, podemos especificarle con la sintaxis:

```
&#número;
```

Donde el número es el código del carácter en decimal. En hexadecimal se puede hacer con:

```
&#número;
```

Ejemplo:

```
<calle>Kantstra&#223;e, Berlín</calle>
```

En el navegador este elemento aparecería como:

```
<calle>Kantstraße, Berlín</calle>
```

El número se puede poner en hexadecimal si se antecede una *x* al nombre, por ejemplo:

```
<calle>Kantstra&#EF;e, Berlín</calle>
```

entidades generales

Se usan como abreviaturas que aparecerán en el documento XML. La razón de su uso es facilitar la escritura de nombres repetitivos (nombres de la empresa, direcciones muy utilizadas,…). La sintaxis para declarar una entidad de este tipo es:

```
<!ENTITY nombre "texto">
```

Para usar en un documento XML la entidad declarada, se usa:

```
&nombre;
```

Ejemplo de declaración de una entidad:

```
<!ENTITY mayor "Calle Mayor Principal" >
```

uso en un documento XML:

```
<dirección>&mayor; 18</dirección>
```

La dirección indicada es *Calle Mayor Principal 18*.

Incluso se pueden indicar símbolos que no son PCDATA al definir entidades:

```
<!ENTITY negCursiva "<strong><em></em></strong>">
```

El documento XML que utilice dicha entidad incluirá todos los símbolos (y por lo tanto estará especificando etiquetas en el código).

Un uso muy interesante es usar entidades que hacen referencia a archivos externos (mediante su dirección URL), por ejemplo:

```
<!ENTITY direcciónCompleta SYSTEM "direccion.txt" >
```

Es la palabra SYSTEM la que indica que la entidad no es un texto sino que es el contenido de un archivo. El uso de la entidad *&direcciónCompleta;* en un documento XML provocará que en dicho documento se añada el contenido del archivo *dirección.txt* (en la posición exacta en la que esté colocada la referencia a la entidad).

entidades de parámetros

Sólo se pueden utilizar dentro del DTD (no en el documento XML). Su uso más habitual es construir DTD utilizando las entidades definidas a fin de ahorrar trabajo al crear el propio DTD.

Su uso es similar a las entidades generales sólo que utilizan el símbolo % en lugar del símbolo &. Al igual que las generales deben de ser declaradas antes de poder usarse:

```
<!ENTITY % mayor "Calle Mayor Principal" >
```

Y su uso (dentro del DTD), por ejemplo:

```
<!ATTLIST persona dirección CDATA "%mayor;">
```

En este caso las comillas dobles son obligatorias porque los valores por defecto van entrecomillados (como se ha visto anteriormente).

Las entidades de parámetros pueden utilizar archivos externos, ejemplo de DTD:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ENTITY % directorio SYSTEM "directorio.dtd" >
%directorio;
<!ELEMENT empresa (razónSocial, dirección) >
<!ELEMENT razónSocial (#PCDATA) >
```

De esta forma se construye un DTD con el contenido ya especificado en otro DTD. En el ejemplo las *empresas* constan de elementos *razónSocial* y de *directorio*. El elemento directorio no se define, sino que su descripción está especificada en *directorio.dtd*.

No obstante el uso más habitual es definir una entidad para utilizar código común en el propio DTD, por ejemplo supongamos que dos elementos, profesor y alumno comparten atributos comunes. Entonces este código simplifica la definición de los atributos de esos elementos:

(2.3) Validación por XML Schema

(2.3.1) XML Schema y DTD

Los DTD en realidad son una reminiscencia del lenguaje SGML que permite un mecanismo de validación para XML al ser éste un subconjunto de SGML.

Por ello, se planteó en torno a 1999 una sintaxis que permitiera más capacidades de validación y que fuera más coherente con el lenguaje XML. Hoy en día es la arquitectura predominante para la validación, ya que se usa en XQuery y sobre todo en los servicios Web.

ventajas de los esquemas

- La sintaxis es XML, por lo que son analizables como cualquier otro documento XML
- Soportan íntegramente los espacios de nombres
- Permiten validaciones de datos avanzadas
- Proporcionan una mayor facilidad para crear validaciones complejas y reutilizables
- Soportan conceptos avanzados como herencia y sustitución de tipos

desventajas de los esquemas

- Son más complejas de entender que las DTD
- Presentan más incompatibilidades con software que las DTD
- No permiten definir entidades
- Tecnologías como SAX o DOM tienen utilidades especiales para las DTD, pero no para los esquemas

(2.3.2) estructura de los esquemas

Un esquema es un documento XML al que se le coloca la extensión xsd. Al ser un archivo XML tiene la estructura habitual de todo documento XML con la obligación de que el elemento raíz se llame schema.

etiqueta schema

La etiqueta schema identifica la raíz de un documento XML Schema. En esta etiqueta se declara el espacio de nombres estándar que utilizan los esquemas (y que permite diferenciar las etiquetas XML del esquema, respecto a las del documento XML), el cual se puede definir como el espacio de nombres por defecto, definir un prefijo xs para él (es la forma habitual) o bien definir un prefijo xsd. Es decir estas tres posibilidades:

- `<schema xmlns="http://www.w3.org/2001/XMLSchema">`
- `<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">`
- `<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">`

A partir de ahí las etiquetas pertenecientes a XML Schema, usarán el prefijo indicado en su espacio de nombres (normalmente **xs**).

Además en la misma etiqueta se define el espacio de nombres al que se aplica el esquema. Es decir, normalmente un esquema XML se aplica a un espacio de nombres privado, correspondiente a la entidad a la que se quiere aplicar el esquema. Este espacio se puede declarar como espacio de nombres por defecto (es lo habitual) o usar un prefijo; incluso se pueden indicar varios espacios de nombres (sólo uno podrá ser definido por defecto como mucho) a los que aplicar el esquema. Como siempre la etiqueta que declara el espacio de nombres es **xmlns**.

Además el atributo **targetNamespace** permite indicar el espacio de nombres sobre el que se aplica el esquema (si se aplica a varios espacios de nombres, aparecerán separados con espacios), que es decir a qué documentos se aplicará el esquema. Ejemplos de etiquetas schema:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.jorgesanchez.net/doc"
           targetNamespace="http://www.jorgesanchez.net/doc">
```

En el ejemplo anterior, las etiquetas correspondoentes al espacio estándar de XML Schema usarán el prefijo **xs**, mientras que las etiquetas pertenecientes a los documentos XML correspondientes al espacio privado ***jorgesanchez.net/doc*** usarán el espacio de nombres por defecto.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:doc="http://www.jorgesanchez.net/doc"
           targetNamespace="http://www.jorgesanchez.net/doc">
```

En este caso XMLSchema usarán el prefijo **xs**, mientras que las etiquetas definidas en el esquema usarán el prefijo **doc** ya asociado a su espacio por defecto:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
   xmlns:doc="http://www.jorgesanchez.net/doc"
   xmlns:img="http://www.jorgesanchez.net/img"
   targetNamespace= "http://www.jorgesanchez.net/doc
                     "http://www.jorgesanchez.net/img ">
```

En este caso las etiquetas XMLSchema usarán el espacio por defecto de nombres; mientras que se definen dos espacios de nombres asociados a los prefijos ***doc*** e ***img***, ambos también indicados en el **targetNamespace**; eso indica que el esquema se aplicará a documentos pertenecientes a ambos espacios de nombres.

La razón de repetir la URI de los espacios de nombres reside en que los atributos **xmlns** y **targetNamespace** no sirven para lo mismo; el primero declara espacios de nombre y el segundo sirve para indicar a qué tipo de documentos se aplicarán las reglas del esquema. Aunque en la práctica ciertamente el contenido de ambos atributos se refiera a lo mismo.

asociar un esquema a un documento XML

Para que un documento XML siga las reglas definidas en un esquema, no disponemos de etiqueta !DOCTYPE; en su lugar utilizamos atributos especiales en el elemento raíz del documento XML.

Primero, al igual que en el documento XMLSchema, necesitamos definir los dos espacios de nombres, el correspondiente al documento XML (que se suele usar sin abreviatura, es decir como espacio por defecto) y el espacio de nombres de XML Schema (que suele utilizar el prefijo **xs**, aunque se puede utilizar otro).

Además es necesario indicar dónde está el archivo XMLSchema que contiene las reglas de validación que se aplican al documento. Esto se hace gracias al atributo llamado **schemaLocation** (perteneciente al espacio de nombres del esquema, por lo que se usa normalmente como **xs:schemaLocation**).

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<documento xmlns="http://www.jorgesanchez.net/doc"
   xmlns:xs="http://w3.org/2001/XMLSchema-instance"
   xs:schemaLocation="esquema.xsd">
  ....
</documento>
```

Se indica el espacio por defecto de nombres en el documento (coincide con el declarado en el propio archivo del esquema), se indica el espacio de nombres correspondiente al esquema (siempre es la misma dirección de Internet) y se asocia a este espacio el prefijo **xs** (se puede elegir otro prefijo, pero no es nada conveniente).

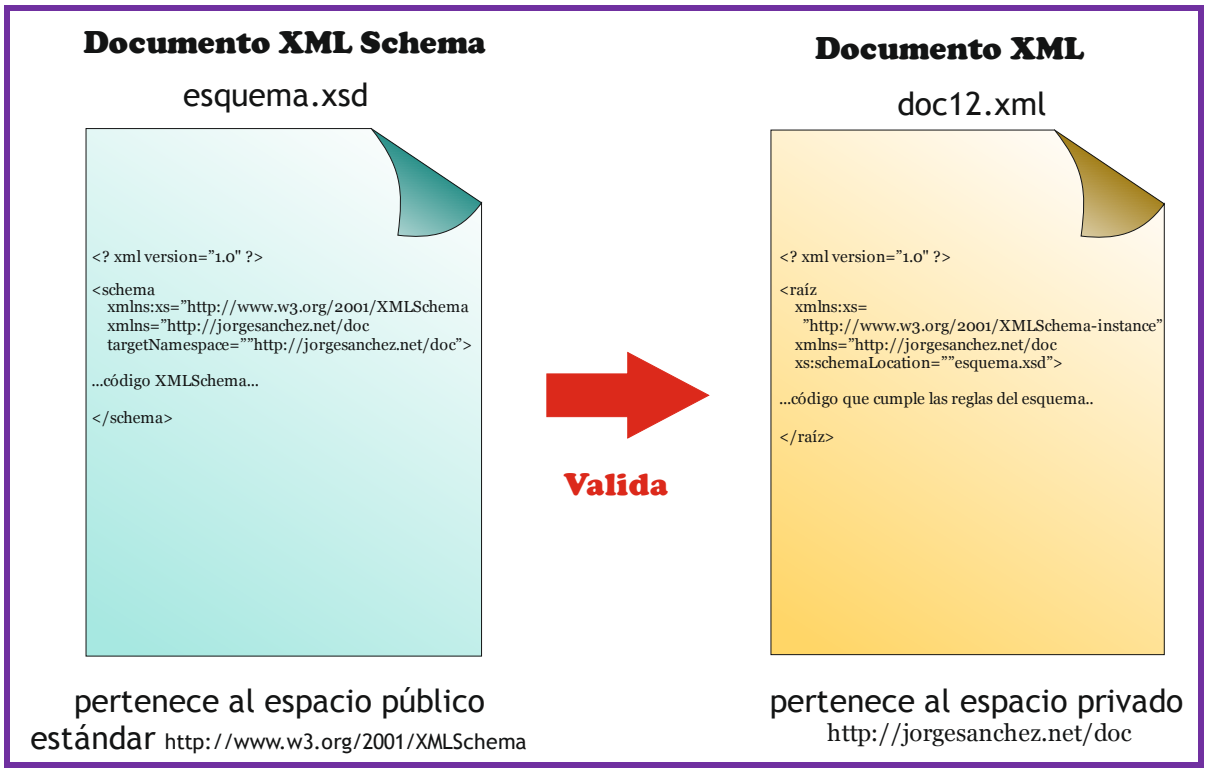

Ilustración 1, Esquema de funcionamiento de los XML Schemas

El atributo **schemaLocation** (acompañado del prefijo asociado al espacio de nombres de XMLSchema) indica la localización del documento XMLSchema que contiene la definición de las reglas a cumplir por el documento. Es un par formado por el espacio de nombres que será validado por el esquema y por la ruta al documento XMLSchema (con extensión **xsd**).

Se pueden indicar varios esquemas de validación, por lo que habría que indicar a qué espacio se aplica cada uno:

```
<?xml version="1.0" encoding="UTF-8"?>
<documento xmlns:doc="http://www.jorgesanchez.net/doc"
           xmlns:img="http://www.jorgesanchez.net/img"
           xmlns:xs="http://w3.org/2001/XMLSchema-instance"
    xs:schemaLocation="http://www.jorgesanchez.net/doc  esquemaDoc.xsd
                       http://www.jorgesanchez.net/img  esquemaImg.xsd
 ....
</documento>
```

La dirección del documento se ha puesto en estilo de ruta relativa, que se calcula a partir de la dirección del documento XML (es decir el documento XMLSchema en el ejemplo estará en la misma carpeta del XML). Pero lo habitual es que a los esquemas se acceda por URL completa; por ejemplo *http://www.obj.com/esq.xsd*

Además podemos indicar un esquema para un documento, pero sin que dicho esquema utilice espacio de nombres. Por ejemplo el esquema podría tener esta cabecera (archivo *esquema1.xsd*):

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="descripción" type="xs:string" />
</xs:schema>
```

El archivo XML Schema anterior no indica ningún espacio de nombres al que aplicarse. Un documento que hiciera referencia al esquema podría ser:

```
<?xml version="1.0" encoding="UTF-8"?>
<descripción xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
  xs:noNamespaceSchemaLocation="esquema1.xsd">
</descripción>
```

El atributo **noNameSchemaLocation** permite indicar un esquema para el documento sin que éste utilice espacio de nombres alguno (lo cual no es nada aconsejable, pero vale para hacer pruebas).

partes de un esquema

- **Elementos**, definidos con etiquetas **xs:element**. Para indicar los elementos permitidos en los documentos que sigan el esquema.
- **Atributos**, etiqueta **xs:attribute**.
- **Tipos simples**, que permiten definir los tipos simples de datos que podrá utilizar el documento XML. Lo hace la etiqueta **xs:simpleType**.
- **Tipos complejos**, mediante la etiqueta **xs:complexType**.

- **Documentación**, información utilizable por aplicaciones que manejen los esquemas. Etiquetas **xs:annotation**, **xs:documentation** y **xs:appInfo**.

componentes locales y globales

El orden de los elementos en un esquema no es significativo, es decir las declaraciones se pueden hacer en cualquier orden. Pero sí que hay que tener en cuenta que dependiendo de dónde coloquemos la definición de los elementos del esquema, varía su ámbito de aplicación. Se distinguen dos posibilidades de declarar elementos:

- En **ámbito global**. Se trata de los elementos del esquema que se coloquen dentro de la etiqueta raíz **schema** y que no están dentro de ninguna otra. Estos elementos se pueden utilizar en cualquier parte del esquema.
- En **ámbito local**. Se trata de elementos definidos dentro de otros elementos. En ese caso se pueden utilizar sólo dentro del elemento en el que están inmersos y no en todo el documento. Es decir si, por ejemplo, si dentro de la definición de un atributo colocamos la definición de un tipo de datos, este tipo de datos sólo se puede utilizar dentro del elemento **xs:attribute** en el que se encuentra la definición del tipo de datos.

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:doc="http://www.jorgesanchez.net/doc"
           targetNamespace="http://www.jorgesanchez.net/doc">
   <xs:element ...>  <!--Definición global
     <xs:simpleType ...> <!--Definición local,
           ...
     </xs:simpleType>
     ...
   </xs:element >
   <xs:simpleType ...> <!--Definición global
           ...
   </xs:simpleType ...>
</xs:schema>
```

El componente local definido sólo se podría utilizar en la zona resaltada. Si fuera global se podría utilizar en todo el documento.

(2.3.3) sintaxis de la definición de un elemento

En XML Schema la definición de un elemento XML se realiza mediante la etiqueta **element**. La sintaxis completa es:

```
<xs:element
    name="nombre del elemento"
    type="tipo global de datos"
    ref="declaración del elemento global"
    id="identificador"
    form="cualificación" <!--qualified o unqualified →
    minOccurs="número mínimo de veces"
    maxOccurs="máximo número de veces"
    default="valor por defecto"
    fixed="valor fijo"
>
```

Al menos hay que indicar el nombre; el tipo de datos también es necesario indicarle casi siempre; el resto de atributos sólo si se necesitan. Por ejemplo se puede definir un elemento como:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="descripción" type="xs:string" />
</xs:schema>
```

El elemento descripción será de tipo string. Los tipos de datos en XML Schema son muchos y además permiten personalizar sus posibilidades para adaptarles a cualquier necesidad.

Poco a poco todos los atributos de la etiqueta **element** se irán describiendo a lo largo del presente documento.

(2.3.4) tipos simples de datos

Tenemos dos posibles variantes:

- **Primitivos**. Los tipos más básicos de XML, están ya definidos por el propio lenguaje XML.
- **Derivados**. Tipos de datos más complejos creados a partir de los anteriores.

tipos de datos ya implementados

tipos de datos primitivos

Son los tipos básicos de XML. Sirven para formar los tipos derivados y tipos más complejos.

Para usarlos basta indicarlos en el atributo **type** de una etiqueta **element** o **attribute** (que son las que permiten crear elementos y atributos). De esta forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="descripción" type="xs:string"/>
</xs:schema>
```

Él código remarcado es el que indica que en los documentos XML basados en esta plantilla XMLSchema, habrá un elemento llamado ***descripción*** que contendrá datos de tipo ***string***. El hecho de que se use el prefijo **xs** (es decir ***xs:string***) es para indicar que es un tipo XML y por lo tanto usa el prefijo designado al espacio de nombres de XMLSchema. Los tipos básicos son:

textos

- **string**. Representan textos con cualquier contenido excepto los símbolos <, > & y las comillas para los que se usará la entidad correspondiente.

lógicos

- **boolean**. Sólo puede contener los valores verdadero o falso, escritos como **true** o **false**, o como **1** (verdadero) ó **0** (falso)

números

- **integer**. Permite utilizar números enteros.
- **float**. Permite utilizar números decimales en formato de coma flotante de precisión simple. El separador decimal es el punto. Ejemplos de valores: ***1, 2 3.4, -123,-124.76, 1E+3*** (significa 1000), ***1.45E+4*** (significa 14500), ***1.45E-4*** (significa 0,000145, es decir $1{,}45 \cdot 10^{-4}$), ***INF*** (infinito), ***-INF*** (menos infinito), ***NaN*** (significa ***Not a Number***, no es número, para números no válidos)
- **double**. Números decimales en formato de coma flotante de precisión doble. Es decir el mismo tipo de números pero con mejor precisión.
- **decimal**. Representa números decimales en coma fija. Ocupan más internamente pero se representan de forma exacta. No admite los valores **INF**, **NaN** ni tampoco el formato científico.
- **hexBinary**. Representa números binarios codificados en notación hexadecimal.
- **base64Binary**. Representa números binarios (usando base 64).

fechas

Se basan en la especificación **ISO 8601**.

- **duration**. Representa duraciones de tiempo en formato **ISO 8601** (sección 5.3) **P**n**Y**n**M**n**D**T**h**m**s** Donde todos los signos ***n*** son números. Ejemplos de valores:
  - ***P1Y*** significa ***Un año***.
  - ***P1Y2M*** significa ***un año y dos meses***.
  - ***P1Y2M3D*** significa ***un año y dos meses y tres días***
  - ***P3D*** significa ***tres días***
  - ***P1Y2M3DT12H30M40.5S*** significa ***un año, dos meses, tres días, doce horas treinta minutos y cuarenta segundosy medio***
  - ***PT12H30M40.5S*** ***doce horas treinta minutos y cuarenta segundosy medio***

Como se observa la **P** es obligatoria y la **T** sirve para separar los valores de fecha de los valores hora.

- **dateTime**. Representa fechas según el formato **ISO 8601** (sección 5.4). El formato es **yyyy-mm-ddThh:mm:ss** , por ejemplo ***1998-07-12T16:30:00.000*** (12 de julio de 1998 a las 16:30). La **T** es obligatoria para separar la fecha de la hora.
- **time**. Representa horas en el formato ***hh:mm:ss***
- **date**. Representa fecha en formato ***yyyy-mm-dd***
- **gYearMonth**. Representa un mes y un año en formato ***yyyy-mm***
- **gYear**. Representa un año usando cuatro cifras.
- **gMonthDay**. Representa un mes y un día en formato ***--mm-dd***
- **gDay**. Representa un día. Hay que hacerlo indicando tres guiones por delante (por ejemplo ***---12***)
- **gMonth**. Representa un mes en formato ***--mm***, por ejemplo **--05**

especiales

- **anyURI**. Representa una dirección URI. Por ejemplo http://www.jorgesanchez.net. La URI puede ser absoluta o relativa.
- **QName**. Nombre cualificado. Representa un nombre XML válido para identificar nombres incluyendo el prefijo de espacio de nombres. Por ejemplo ***doc:cabecera***
- **Notation**. Representa notaciones de estilo **NOTATION** XML 1.0 segunda edición. Sólo se debe utilizar para crear tipos de datos derivados de éste
- **anyType**. No restringe el contenido en modo alguno.

tipos de datos derivados

Son datos que se han definido a partir de los anteriores, pero forman parte de XMLSchema, es decir que en la práctica se usan igual que los anteriores (al igual que en los primitivos, cuando se usan en un esquema hay que añadir el prefijo del espacio de nombres del esquema, por ejemplo ***xs:normalizedString***).

textos

- **normalizedString**. Se basa en el tipo **string**. Texto donde los caracteres de retorno de línea, tabulador y retorno de carro se convierten a espacios antes de procesar el esquema.
- **token**. Se basa en el anterior. Textos en los que no hay más de un espacio en blanco seguido, ni tabuladores ni saltos de línea; en todos esos casos se convierte el texto a un único espacio.
- **language**. Texto que contiene el nombre de un lenguaje según lo definido en la especificación oficial **RFC 1766.** Son los posibles valores de los atributos **xml:lang** de XML 1.0 que coinciden con el formato normalizado de lenguajes habitual en las páginas web; por ejemplo el español se codifica con ***es*** (a veces con ***es-ES***), el catalán ***ca***, el gallego ***gl***, el portugués ***pt***, el euskera ***eu***, el inglés ***en***, alemán ***de*** y francés ***fr***.

- **Name**. Sólo admite nombres compatibles con la forma de poner nombres de XML. Admite los dos puntos pero para manejar nombres con prefijo (nombres cualificados) el tipo idóneo es **QName**.
- **NCName**. Nombres, basado en **Name**, pero sin admitir los dos puntos de los prefijos de espacios de nombres.

números

- **integer**. Basado en **number**. Representa números enteros tanto positivos como negativos
- **nonPositiveInteger**. Basado en **integer**. Representa números enteros que no son positivos (es decir cero y negativos).
- **negativeInteger**. Basado en **integer**. Representa números enteros negativos (no vale el cero).
- **nonNegativeInteger**. Basado en **integer**. Representa números enteros que no son negativos (es decir cero y positivos).
- **positiveInteger**. Representa números enteros positivos puros (no vale el cero).
- **long**. Basado en **integer**. Representa números enteros de alto rango (de -*9223372036854775898* a *9223372036854775897*). Los **integer** representan números aún más largos
- **unsignedLong**. Basado en **long**. Representa números enteros de alto rango pero usando sólo los positivos y el cero.
- **int**. Representa números enteros de medio rango (de -2147483648 a 2147483647).
- **unsignedInt**. Representa números enteros de medio rango pero usando sólo los positivos y el cero.
- **short**. Representa números enteros de bajo rango (de -32768 a 32767).
- **unsignedShort**. Representa números enteros de medio rango pero usando sólo la parte positiva, de cero a 65535
- **byte**. Representa números enteros pequeños (de -128 a 127).
- **unsignedByte**. Representa números enteros pequeños positivos, de cero a 255

equivalentes a atributos DTD

Están basados en los tipos XML 1.0 y sólo pueden utilizarse en atributos (para mantener la compatibilidad con DTD)

- **ID, IDREF e IDREFS**. Equivalente a los atributos del mismo tipo de XML 1.0 (tienen el mismo significado que en las DTD). Derivan de **NCName**
- **ENTITY, ENTITIES**. Equivalente a los atributos XML 1.0 del mismo nombre. Derivan de **NCName**,
- **NMTOKEN, NMTOKENS**. Permiten indicar textos compatibles con los nombres XML. Derivan de **NCName**.
- **NOTATION**. Es un tipo pensado para hacer anotaciones, su funcionamiento es peculiar y no está pensado para ser usado como tipo básico, sino como base para crear tipos personales.

uso de tipos de datos en componentes de XMLSchema

Para que un determinado componente del esquema (como un elemento o un atributo) use uno de los tipos, se dispone del atributo **type** al que se le indicaría el tipo de datos. Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:doc="http://www.jorgesanchez.net/doc"
  targetNamespace="http://www.jorgesanchez.net/doc">
  <xs:element name="documento" type="xs:NCName"/>
  <xs:element name="contenido" type="xs:string" />
  <xs:element name="precio" type="xs:decimal" />
</xs:schema>
```

En el ejemplo se trata de las primeras líneas de un código XMLSchema donde se definen tres elementos, cada uno de los cuales con un tipo distinto. Observar el uso del prefijo *xs* asignado al espacio de nombres de XMLSchema antes del nombre del tipo de datos.

definir tipos simples personales

La sintaxis general es:

```
<xs:simpleType name="nombre">
    ...definición del tipo....
</xs:simpleType>
```

definir tipos por unión

Se trata de utilizar dentro del tipo de datos una etiqueta llamada **union** que permite unir las definiciones de dos tipos de datos. Por ejemplo:

```
<xs:simpleType name="gMonthC">
  <xs:union memberTypes="xs:gMonth xs:gMonthDay" />
</xs:simpleType>
```

Cuando a cualquier elemento del esquema se le asigne el tipo *gMonthC*, se podrán especificar datos en formato *gMonth* y en formato *gMonthDay*.

establecer tipos simples por restricción

Permiten establecer reglas complejas que deben de cumplir los datos. En este caso dentro de la etiqueta **simpleType** se indica una etiqueta **restriction**, dentro de la cual se establecen las posibles restricciones. Sintaxis:

```
<xs:simpleType name="nombre">
    <xs:restriction base="tipo">
            ...definición de la restricción....
    <xs:restriction>
</xs:simpleType>
```

El atributo **base** sirve para indicar en qué tipo nos basamos al definir la restricción (es decir de qué tipo estamos creando este derivado). El apartado restriction pueden tener numerosas etiquetas que permiten establecer las restricciones deseadas al tipo.

Las etiquetas interiores a **restriction** disponen de un atributo llamado **fixed** que sólo puede valer verdadero (**true**) o falso (**false**). En caso de que sea verdadero, ningún tipo derivado del definido puede modificar la propiedad establecida; es decir, si establecemos **minLength** (tamaño mínimo) con valor *ocho* (propiedad **value**) y **fixed="true"**, ningún tipo derivado del definido podrá definir que el tamaño mínimo sea inferior a 8 caracteres. Es un atributo de uso opcional.

Las posibles restricciones que se pueden establecer son:

- **Tamaños de texto**. Indica tamaños máximos y mínimos que debe de tener el texto. Ejemplo:
  - **minLength**. Indica el mínimo número de caracteres. Eso lo hace mediante el atributo **value,** en el que se indica un número con el tamaño mínimo que deseamos.
  - **maxLength**. Indica un tamaño máximo de caracteres o de dígitos numéricos. Usa el mismo atributo **value**.
  - **length**. Indica un tamaño fijo de caracteres para el tipo. Es decir si indicados *length* con *value="9"* el texto deberá tener exactamente nueve caracteres.

  Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:doc="http://www.jorgesanchez.net/doc"
           targetNamespace="http://www.jorgesanchez.net/doc">
  <xs:simpleType name="nombresTipo">
    <xs:restriction base="xs:normalizedString">
      <xs:maxLength value="15" />
      <xs:minLength value="4" />
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="título" type="doc:nombresTipo" />
```

  En el ejemplo, se define el tipo persona *nombresTipo* que permite textos entre 4 y 15 caracteres. El hecho de declararle como derivado de **normalizedString** permite restringir que no considere en los textos más de un espacio seguido ni los tabuladores ni saltos de línea (de otro modo casi siempre se superaría el mínimo de cuatro, quizá incluso fuera mejor derivar de *NCNames* que es aún más restrictivo).

- **Dígitos máximos**. Parecido al anterior pero trabajando con números. Indica las posibles cifras que puede tener el número.
  - **totalDigits**. Número máximo de dígitos del número, incluyendo los decimales. El atributo value indica el número máximo deseado
  - **fractionDigits**. Máximo número de decimales que puede tener el número.

```
<xs:simpleType name="tipo1">
  <xs:restriction base="xs:decimal">
    <xs:totalDigits value="6" />
    <xs:fractionDigits value="2" />
  </xs:restriction>
</xs:simpleType>
<xs:element name="para" type="tipo1"  />
```

En el ejemplo, los elementos que utilicen el *tipo1* definido, podrán escribiré números de hasta seis cifras (incluidos

- **Máximos y mínimos numéricos**. Restringe valores numéricos asignando topes a los mismos. Sirve para números y para valores de tiempo o duración. Se hace con:
  - **minExclusive**. Establece un valor mínimo. El valor debe ser mayor que el establecido por la etiqueta a través del atributo **value**.
  - **maxExclusive**. Establece un valor máximo. El valor debe ser menor que el establecido por la etiqueta a través del atributo **value**.
  - **minInclusive**. Establece un valor mínimo. El valor debe ser mayor o igual que el establecido por la etiqueta a través del atributo **value**.
  - **maxInclusive**. Establece un valor máximo. El valor debe ser menor o igual que el establecido por la etiqueta a través del atributo **value**.
- **Espacios en blanco**. Sirve para indicar la política de manejo de espacios en blanco en los textos. La etiqueta que lo controla es **whiteSpace** y tiene tres posibles valores para el atributo **value**:
  - **preserve**. No modificar espacios en blanco, ni tabuladores ni saltos de línea. Es decir se les tendrá en cuenta.
  - **replace**. Cada doble espacio o tabulador o salto de línea se cambia por un espacio (al estilo del tipo predefinido **normalizedString**)
  - **collapse**. Como el anterior, pero además elimina los espacios a izquierda y derecha. Es muy útil para usar en combinación con las propiedades de tamaño de texto vistas anteriormente.

Ejemplo (archivo *prueba.xsd*):

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:simpleType name="tipo1">
      <xs:restriction base="xs:string">
        <xs:whiteSpace value="collapse" />
      </xs:restriction>
    </xs:simpleType>
```

```
    <xs:element name="prueba" type="tipo1" />
</xs:schema>
```

En e archivo anterior, se define un elemento llamado *prueba* que tiene un tipo que colapsa los espacios en blanco y así en un documento XML que aplique este esquema, por ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<prueba xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
  xs:noNamespaceSchemaLocation="prueba.xsd">
  Este es el          texto
                        que deseo

                        probar
</prueba
```

Cuando mostremos el resultado en un programa que aplique el esquema (por ejemplo en las últimas versiones de **Internet Explorer, Firefox, Safari** o **Chrome**), tendremos el contenido de la etiqueta prueba en esta forma:

```
Este es el texto que deseo probar
```

- **Enumeraciones**. Limitan el contenido a una lista de valores. Se realizan con una sucesión de etiquetas **enumeration**. Estas etiquetas se pueden combinar con las propiedades anteriores (aunque no tiene sentido) y no poseen atributo **fixed**. De modo que si un tipo deriva de una enumeración, podrá volver a enumerar para indicar valores válidos y estas deberán estar en la enumeración anterior; una vez más podremos restringir más, pero nunca menos.

  Ejemplo de enumeración:

```
<xs:simpleType name="sexoTipo">
  <xs:restriction base="xs:NCName">
    <xs:enumeration value="Hombre"/>
    <xs:enumeration value="Mujer"/>
  </xs:restriction>
</xs:simpleType>
```

  Mediante este tipo sólo podremos elegir como valores *Hombre* o *Mujer*.

- **Plantillas** (*pattern*). Permite establecer **expresiones regulares;** es decir, un texto con símbolos especiales que permiten establecer expresiones que ha de cumplir el contenido. Las expresiones regulares son bien conocidas por casi todos los programadores y dan una potencia increíble para establcer restricciones de texto avanzadas, lo que las hace muy utilizadas.

  Las plantillas se manejan con etiquetas **pattern** a las que, en el atributo **value**, se indica un texto con símbolos especiales que especifica la expresión a cumplir. Los símbolos que se pueden utilizar son:

Símbolo	Significado
texto tal cual	Hace que sólo se pueda escribir ese texto. Por ejemplo si se indica "*Hombre*", la restricción será escribir como valor posible

Símbolo	Significado
	exactamente el texto *Hombre*.
[xyz]	Permite elegir entre los caracteres *x*, *y* o *z*
[^xyz]	Prohibe usar cualquiera de los caracteres entre corchetes
[a-z]	Vale cualquier carácter de la *a* a la *z*.
^	Inicio de línea
$	Final de línea
+	Repite acepta el carácter precedente una o más veces
?	Acepta el carácter precedente 0 o más veces
*	Acepta el carácter precedente una o más veces
{n}	Acepta exactamente *n* repeticiones del carácter precedente.
{n,}	Acepta al menos *n* repeticiones del carácter precedente.
{n,o}	Acepta entre *n* y *n* repeticiones del carácter precedente.
\s	Permite indicar los caracteres especiales. Por ejemplo \^ representa el carácter circunflejo ^ para que sea tomado como texto y no como código especial.

Por ejemplo la validación para un dato tipo DNI (8 cifras y un número), sería:

```
<xs:simpleType name="dniTipo">
  <xs:restriction base="xs:string">
    <xs:whiteSpace value="collapse" />
    <xs:pattern value="[0-9]{8}[A-Z]" />
  </xs:restriction>
</xs:simpleType>
```

- enumeraciones. Las realiza una etiqueta llamada enumeration que sirve para indicar los posibles valores que puede tomar un componente. Por ejemplo:

```
<xs:simpleType name="diasSemanaTipo">
  <xs:restriction base="xs:string">
    <xs:whiteSpace value="collapse" />
    <xs:enumeration value="Lunes" />
    <xs:enumeration value="Martes" />
    <xs:enumeration value="Miércoles" />
    <xs:enumeration value="Jueves" />
    <xs:enumeration value="Viernes" />
    <xs:enumeration value="Sábado" />
    <xs:enumeration value="Domingo" />
  </xs:restriction>
</xs:simpleType>
```

definir tipos simples por lista

Las listas permiten que un componente tenga como contenido una determinada lista de valores. la construcción de listas indica dos pasos:

(1) Crear un tipo simple de datos cuyo contenido es una etiqueta ***list***, la cual posee el atributo **itemType** para indicar el tipo de elementos de la lista. Simplemente con ello e podría establecer una lista (valores separados por espacios) de valores pertenecientes al tipo indicado.

(2) Crear el tipo ya definitivo de datos que contendrá una etiqueta **restriction** a la cual como tipo base se indica el tipo simple de datos relacionado con la lista. EN la restricción se pueden indicar estas etiquetas usando el atributo **value**:

- **length**. Indica que la lista tendrá un número exacto de valores.
- **minLength.** Indica que la lista tendrá un número mínimo de valores
- **maxLength**. Indica que la lista tendrá un número máximo de valores
- **enumeration**. Posibles valores que puede tener la lista
- **whiteSpace**. Gestión de los espacios en blanco en cada elemento de la lista.
- **pattern**. Expresión regular que debe cumplir cada elemento de la lista.

Es decir, son las etiquetas ya conocidas, pero que ahora se refieren a la lista. Ejemplo:

```
<xs:simpleType name="listaDecimales">
  <xs:list itemType="xs:decimal" />
</xs:simpleType>
<xs:simpleType name="listaNotas">
  <xs:restriction base=" listaDecimales">
    <xs:minLength value="3" />
    <xs:maxLength value="6" />
  </xs:restriction>
</xs:simpleType>
```

Primero se define un tipo de lista (***listaDecimales***) simplemente indicando que será una lista cuyos valores serán números decimales. Después se concreta la lista (***listaNotas***), de modo que ahora se indica que la lista constará de tres a seis números (en el XML cada número irá separado por espacios)

definiciones locales de tipos

Como se comentó anteriormente, los componentes de XML Schema pueden definirse en modo local o global. En modo global cualquier otro componente le puede utilizar, en modo local sólo aquel que contiene la definición. Los datos simples se pueden crear, por ejemplo, en un elemento y así se aplican sólo a dicho elemento. Ejemplo:

```
<xs:element name="elemento1">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:whiteSpace value="collapse" />
      <xs:maxLength value="16" />
    </xs:restriction>
```

```
    </xs:simpleType>
</xs:element>
```

El tipo simple declarado ni siquiera tiene nombre, ya que no se utilizará fuera del elemento *elemento1*.

(2.3.5) atributos

Los atributos se definen parecido a los elementos. Su sintaxis de definición es:

```
<xs:attribute
    name="nombre del elemento"
    type="tipo global de datos"
    ref="declaración del elemento global"
    form="cualificación" <!--qualified o unqualified →
    id="identificador"
    default="valor por defecto"
    fixed="valor fijo"
    use="uso" <!-- prohibited, optional o required →
>
```

Los atributos sólo pueden asignarse a los elementos mediante tipos complejos de datos. El uso hace referencia a si al atributo no se le puede asignar ningún valor en el documento XML (valor **prohibited**), si su uso es opcional (valor **optional**) o si es obligatorio darle valor.

(2.3.6) tipos compuestos

Los tipos de datos del apartado anterior sólo sirven para indicar contenidos simples (que contienen sólo información en el interior) de elementos o bien para indicar posibles valores a los atributos. Los tipos compuestos permiten definir contenidos más complejos. Puesto que lo normal es que los elementos de un documento XML puedan contener otros elementos y por supuesto atributos, es lógico que la mayoría de elementos indiquen mediante tipos compuestos su contenido. Los datos simples son apropiados para indicar el tipo de contenido de los atributos o bien para indicar el contenido de los elementos simples (lo que en DTD serían elementos sólo con contenido **#PCDATA**).

Al igual que los datos simples, los compuestos pueden ser globales o locales. En el caso de ser locales no se indica un nombre (atributo **name**) y entonces sólo se podrán utilizar para el elemento en el que se definieron. Los globales se pueden utilizar para distintos elementos y por lo tanto requieren que se indique su nombre.

Los tipos compuestos se definen con la etiqueta **complexType**. En esa etiqueta podemos utilizar diferentes modelos de definición de contenidos.

En realidad los elementos XML desde el punto de vista de la sintaxis XML Schema pueden tener cuatro tipos de contenido:

- **Contenido simple** (*Simple Content*). Sólo admite texto en su interior y no otros elementos (son los elementos definidos en DTD como **(#PCDATA)**)
- **Vacíos** (*Empty*). No pueden contener ni más elementos dentro ni texto.

- **Contenido compuesto** (*Complex Content*). Pueden contener otros elementos, pero no PCDATA.
- **Contenido mixto** (*Mixed Content*). Pueden contener tanto texto como más elementos.

La sintaxis de la etiqueta complexType admite señalar contenidos simples y compuestos. Para los vacíos y mixtos hay que utilizar (se verá más adelante) atributos especiales en la etiqueta element. Sin indicar nada especial, compexType parte de que estamos definiendo contenidos complejos (es decir, elementos que contienen más elementos).

Los elementos pueden contener atributos, más adelante se indica la forma de incorporarlos.

elementos vacíos

Para indicar que un elemento es vacío basta con no indicar valores e indicar el nombre del elemento sin indicar tipo de datos alguno. Ejemplo:

```
<xs:element name="casado" >
  <xs:complexType>
    <xs:attribute name="valor" type="xs:string" />
  </xs:complexType>
</xs:element>
```

En este caso el elemento caso sólo dispone de un atributo llamado valor, no será posible meter ningún contenido en la etiqueta de casado. Si no deseamos atributos (aunque es muy extraño), entonces simplemente no habrá etiquetas attribute (pero sí todas las demás).

definición de contenidos con texto

Se trata de usar la etiqueta <xs:element al estilo que se ha usado en los ejemplos. Es decir se indica el nombre y el tipo de datos e incluso se pueden indicar atributos dentro del apartado complexType, pero nunca se ponen elementos dentro de este elemento y así sólo se admitirá por contenido el texto (tipo PCDATA).

Ejemplo:

```
<xs:element name="nombre" type="xs:string" />
```

definición de contenidos con texto y atributos

En este caso se indica que el elemento posee contenido simple en la etiqueta complexType dentro de la cual se indican los atributos, mientras que es la etiqueta del contenido simple (simpleContent) la que poseerá el tipo de datos para el contenido del elemento.

Hay dos formas de indicar contenido simple: por extensión (mediante etiqueta extension) y por restricción (etiqueta restriction).

Los atributos se deben indicar en el apartado extension que es el encargado de indicar los tipos . Ejemplo:

```
<xs:element name="documento">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="idioma" type="xs:string" />
      </xs:extension>
```

```
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

En el ejemplo, documento es un elemento de tipo string (texto) que contiene un atributo llamado idioma (también string). Es un poco enrevesado, pero es necesario hacerlo así.

definición de contenidos compuestos

Como se ha comentado antes, los contenidos compuestos se refieren a los elementos que contienen otros elementos (pero nunca texto libre). Hay tres posibles tipos de elementos a contener: **secuencias**, **elecciones** y contenidos libres (**all**). Además se pueden incorporar atributos.

secuencias

Dentro de un elemento es habitual indicar su contenido como una secuencia de elementos. Esto se permite con la etiqueta **sequence**, dentro de la cual se añaden etiquetas **element** para indicar los elementos que entran en la secuencia. Ejemplo:

```
<xs:simpleType name="emailT">
  <xs:restriction base="xs:string">
    <xs:pattern value="[A-Za-z]{3,}@.{3,}" />
  </xs:restriction>
</xs:simpleType>
<xs:element name="email">
  <xs:complexType >
    <xs:sequence>
      <xs:element name="remite" type="emailT"  />
      <xs:element name="para" type="emailT" minOccurs="1"
                  maxOccurs="unbounded" />
      <xs:element name="CC" type="emailT" minOccurs="0"
                  maxOccurs="unbounded" />
      <xs:element name="CCO" type="emailT" minOccurs="0"
                  maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

En el ejemplo, el elemento email está compuesto de cuatro elementos. El ***remite*** (que tiene obligatoriamente que aparecer una vez), el ***para*** que aparecerá al menos una vez y que puede aparecer tantas veces como se desee y los apartados opcionales ***CC*** y ***CCO*** que pueden aparecer repetidos.

La etiqueta **sequence** posee los atributos **minOccurs** y **maxOccurs** para indicar que el bloque de la secuencia se puede repetir.

elecciones

Sirven para permitir elegir uno de entre varios elementos. Su funcionamiento es el mismo que en las secuencias, pero en este caso se utiliza una etiqueta llamada **choice**.

Ejemplo:

```
<xs:element name="identificación">
<xs:complexType>
  <xs:choice>
    <xs:element name="firma" type="xs:NCName"/>
    <xs:element name="código" type="xs:NCName" />
  </xs:choice>
</xs:complexType>
```

En el ejemplo, el elemento *identificación*, consta de dos posibles elementos *firma* y *código* de los que sólo se podrá incluir uno.

La etiqueta choice también posee los atributos minOccurs y maxOccurs.

etiqueta all

Se trata de una posibilidad similar a choice y sequence que se utiliza de la misma forma y que tiene como diferencia principal que los elementos que contiene pueden aparecer cero o una vez y además en el orden que quieran. Ejemplo:

```
<xs:element name="identificación">
<xs:complexType>
  <xs:all>
    <xs:element name="firma" type="xs:NCName"/>
    <xs:element name="código" type="xs:NCName" />
  </xs:all>
</xs:complexType>
```

En este caso la *firma* y el *código* pueden aparecer o no, aparecer los dos e incluso el orden será indiferente. Es una etiqueta muy potente que ahorra mucho trabajo.

Esta etiqueta tiene los atributos minOccurs y maxOccurs, pero sólo se puede indicar como valores cero o uno.

mezcla de elementos

A veces los contenidos de un documento XML son extremadamente complejos y por eso se permite en los esquema colocar etiquetas choice dentro de etiquetas sequence y viceversa. Y lo mismo ocurre con las etiquetas all. Estas posibilidades permiten crear cualquier tipo de esquema por complejo que resulte. Ejemplo:

```
<xs:element name="correo">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="remite" type="xs:string"/>
        <xs:element name="para" type="xs:string"
                minOccurs="1" maxOccurs="unbounded"/>
        <xs:choice>
          <xs:element name="cc" type="xs:string"
                minOccurs="1" maxOccurs="unbounded" />
          <xs:element name="cco" type="xs:string"
                minOccurs="1" maxOccurs="unbounded" />
        </xs:choice>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

En el ejemplo, el elemento *correo* consta de tres elementos: *remite*, *para* y un tercero que puede ser *cc* o *cco*.

añadir atributos

En los apartados complexType, los atributos del elemento se definen al final del apartado complexType (justo antes de cerrarle). Ejemplo:

```
<xs:element name="persona">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nombre" type="xs:string" />
      <xs:element name="apellidos" type="xs:string" />
    </xs:sequence>
    <xs:attribute name="sexo">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:enumeration value="Hombre" />
          <xs:enumeration value="Mujer" />
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="fechaNacimiento"
                  type="xs:date" use="required" />
  </xs:complexType>
</xs:element>
```

Las personas contienen dos elementos en secuencia (*nombre* y *apellidos*) y dos atributos: uno opcional (*sexo*) que sólo pueden tomar los valores *Hombre* o *Mujer* y uno obligatorio para la fecha de nacimiento

contenidos mixtos

Es el caso más complejo. Se trata de elementos que contienen otros elementos y además texto (e incluso atributos). Para permitir esta posibilidad hay que marcar el atributo mixed de la etiqueta complexType a true. Ejemplo:

```
<xs:element name="documento">
  <xs:complexType mixed="true">
    <xs:choice>
      <xs:element name="negrita" type="xs:string"
          minOccurs="0" maxOccurs="unbounded"/>
    </xs:choice>
    <xs:attribute name="lenguaje" type="xs:language" />
  </xs:complexType>
</xs:element>
```

Lo malo es que no se puede controlar el tipo de datos del texto interior. Los elementos se controlan completamente, pero el texto no.

uso de elementos y atributos globales

Se trata de definir un elemento o un atributo para ser reutilizado en diferentes partes del documento. La forma de utilizarlos es:

- Definirlos en la zona global (es decir, no definirlos dentro de ningún otro componente)

- En el caso de los elementos no se pueden indicar los atributos **minOccurs** y **maxOccurs** para indicar la cardinalidad, porque sólo tiene sentido dentro de otro elemento.
- En el caso de los atributos no se puede utilizar el atributo **use**, que indica la obligatoriedad de uso del atributo.
- Una vez definidos, donde se quieran reutilizar se define el elemento o atributo dentro del componente en el que se quiere colocar y no se le da nombre, sino que se usa el atributo **ref** para indicar el nombre del elemento o atributo global.

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="email">
    <xs:simpleType>
      <xs:restriction base="xs:string">
        <xs:pattern value="[A-Za-z]{3,}@.{3,}"></xs:pattern>
      </xs:restriction>
    </xs:simpleType>
  </xs:element>

  <xs:element name="trabajador">
    <xs:complexType>
      <xs:sequence>
        <xs:element  name="nombre" type="xs:string" />
        <xs:element  name="apellidos" type="xs:string" />
        <xs:element  ref="email" minOccurs="1"
                     maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

En este caso se ha definido un elemento global llamado email que representa una dirección de email (dentro se define un tipo simple para restringir la forma en la que se debería rellenar un email) y que después el elemento ***trabajador*** la reutiliza para indicar los correos electrónicos de un trabajador. Otros elementos podrían reutilizar esta misma definición.

(2.3.7) grupos de elementos

La etiqueta **group** permite realizar grupos de elementos y eso permite organizarse mejor a la hora de crear un esquema. Dentro de cada grupo podemos utilizar etiquetas **sequence, choice** y **all** de la misma forma que la vista anteriormente y así después utilizar el grupo en la forma deseada. Muchas veces los grupos se definen de forma global y así se pueden utilizar en distintos elementos; pero es posible definirles localmente (suele tener menos interés hacerlo en local).

Los grupos cuando se definen de forma global requieren indicar un nombre para ellos (si se definen de forma local no). Cuando un elemento desea incorporar un grupo global, utiliza la etiqueta **group** y con el atributo **ref** indicaría el nombre del elemento global

definido anteriormente. Pueden contener los atributos **minOccurs**, **maxOccurs** para indicar las veces que puede repetirse el grupo en el elemento que le contiene.

Ejemplo:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:group name="seccionesCorreo">
    <xs:sequence>
      <xs:element  name="remite" type="xs:string"/>
      <xs:element  name="para" type="xs:string" minOccurs="1"
                   maxOccurs="unbounded"/>
      <xs:choice>
        <xs:element name="cc" type="xs:string" minOccurs="1"
              maxOccurs="unbounded" />
        <xs:element name="cco" type="xs:string" minOccurs="1"
              maxOccurs="unbounded" />
      </xs:choice>
    </xs:sequence>
  </xs:group>

  <xs:element name="correo">
    <xs:complexType>
      <xs:sequence>
        <xs:group ref="seccionesCorreo" />
        <xs:element name="contenido" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

(2.3.8) grupos de atributos

La idea es la misma que con los grupos de elementos: facilitar la escritura del código del esquema. Se trata de definir (lo lógico es de forma global) una serie de atributos que utilizan diferentes elementos y así mejorar el mantenimiento del esquema XML.

Los grupos de atributos se declaran con **attributeGroup**, al definir se utiliza el nombre y al usarle en un elemento se hace referencia a dicho nombre mediante la etiqueta **ref**.

Ejemplo:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:attributeGroup name="infoDoc">
    <xs:attribute name="lenguaje" type="xs:language"
          use="required"/>
    <xs:attribute name="tamaño">
      <xs:simpleType>
        <xs:restriction base="xs:positiveInteger">
          <xs:minExclusive value="20" />
          <xs:maxExclusive value="200" />
        </xs:restriction>
```

```
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="título" type="xs:string" use="required"/>
    </xs:attributeGroup>

    <xs:element name="documento">
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base="xs:string">
            <xs:attributeGroup ref="infoDoc" />
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>

    <xs:element name="documentoRef">
      <xs:complexType>
        <xs:attributeGroup ref="infoDoc" />
        <xs:attribute name="referencia" type="xs:positiveInteger"
                      use="required" />
      </xs:complexType>
    </xs:element>

    <xs:element name="documentos">
      <xs:complexType>
        <xs:choice minOccurs="1" maxOccurs="unbounded">
          <xs:element ref="documento" />
          <xs:element ref="documentoRef" />
        </xs:choice>
      </xs:complexType>
    </xs:element>
</xs:schema>
```

Se define en el ejemplo un grupo de atributos globales que se llama *infoDoc* y que está formado por tres atributos (*lenguaje*, *tamaño* y *título)* que se definen completamente en el grupo. Después dos elementos (documento y documentoRef) utilizan el grupo de atributos como si fuera un solo atributo. De modo que un XML basado en este esquema y válido podría ser:

```
<?xml version="1.0" encoding="UTF-8"?>
<documentos
    xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
    xs:noNamespaceSchemaLocation="correo.xsd" >
  <documento lenguaje="es" tamaño="123" título="Informe 32" />
  <documento lenguaje="es" tamaño="34" título="Alta 34" />
  <documentoRef      lenguaje="es" referencia="3234"
                     título="Informe 56"/>
</documentos>
```